Available online www.jsaer.com

***Journal of Scientific and Engineering Research*, 2021, 8(1):219-224**

Research Article

ISSN: 2394-2630
CODEN(USA): JSERBR

Auto-stopping in Kubernetes: Optimizing Resource Utilization through Automated Stop and Start

Venkata Sasidhar Kanumuri

Email id: sasinrrt@gmail.com

Abstract Cloud environments demand constant optimization to balance performance and cost-effectiveness. Idle resource consumption is a major hurdle due to cloud resources' on-demand nature. Traditional scaling mechanisms like Horizontal Pod Autoscaler (HPA) help, but they often maintain minimum resource allocation even during low activity. Auto-stopping emerges as a compelling solution, automating the deactivation of idle resources during periods of low demand. This approach leads to significant cost savings without compromising application availability, making it a valuable tool for optimizing resource utilization within Kubernetes deployments.

Auto-stopping offers many benefits. It eliminates idle resource charges, substantially reducing costs, particularly for non-production and sporadically used environments. Studies suggest potential savings ranging from 30% to 75%. Additionally, auto-stopping optimizes overall cluster resource utilization by intelligently suspending unnecessary resources. This frees up resources for active workloads and minimizes environmental impact through reduced energy consumption. Furthermore, auto-stopping automates resource shutdowns and startups, streamlining management tasks and reducing operational overhead, especially for large-scale deployments with fluctuating workloads.

Keywords: auto-stopping, scheduled stop and start, containers cost optimization, Kubernetes, k8s resource optimization, cost management, KEDA.sh, clusters, HPA, benefits, K8s cron jobs

Introduction

Cloud settings are dynamic, meaning resource use must be constantly optimized to maintain performance and cost-effectiveness. Nevertheless, several issues make this goal difficult to achieve. The first is that idle resource consumption is a risk inherent in the on-demand nature of cloud resources. Not allotted resources will be used to their full potential when workloads are not at their highest level, which results in wasteful spending. It can be challenging to precisely forecast variations in workload, especially in settings where traffic patterns are erratic or bursty. Conventional scaling techniques like Horizontal Pod Autoscaler (HPA) can partially address the issue by dynamically modifying resource allocation in response to demand. Even in times of low activity, HPA frequently keeps a minimal number of replicates, which presents limits. This emphasizes the need for more granular control over resource allocation, particularly for non-critical workloads or during off-peak hours.

Auto-stopping emerges as a compelling solution to these challenges. It is an automated technique for deactivating idle resources during periods of low demand. Automatically suspending unnecessary resources helps optimize resource utilization and minimize cloud infrastructure costs. This approach complements scaling mechanisms like HPA by enabling complete resource shutdown when workloads are minimal. The following sections will delve deeper into the benefits and approaches for implementing auto-stopping within Kubernetes deployments.

In resource-intensive cloud environments, idle Kubernetes resources represent a significant cost burden. Auto-stopping, an automated deactivation technique, tackles this challenge by intelligently suspending unnecessary

resources during periods of low demand. This section delves into the benefits and approaches of auto-stopping within Kubernetes deployments.

Need for Auto-Stopping in Resource Optimization: Minimizing Costs without Impacting Availability

In the current cloud-native environment, where infrastructure expenses remain a major concern, making the most use of available resources is essential. While autoscaling adapts infrastructure dynamically to suit varying workloads, auto-stopping automatically deactivates idle resources during times of low demand, hence enhancing resource efficiency.

[1]. Cost Savings for Underutilized Resources: During non-business hours or periods of minimal activity, a significant portion of Dev/Test environments lie idle, incurring unnecessary expenditure. Auto-stopping tackles this challenge by intelligently shutting down these idle resources, leading to potential cost savings of 33% or more during weekdays and even higher on weekends.

[2]. Streamlined Management and Efficiency: Manual handling of resource shutdowns and startups, particularly in large-scale deployments, can be cumbersome and error-prone. Auto-stopping automates this process, ensuring resource availability during scheduled working hours and eliminating the need for manual intervention. This frees up valuable IT resources for higher-level tasks, boosting operational efficiency.

[3]. Environmental Impact Reduction: Auto-stopping immediately results in lower energy consumption and a smaller carbon footprint by reducing idle resource consumption. This aligns with the growing ecological worries and enables businesses to demonstrate their dedication to sustainable operations.

[4]. Flexibility and Control: Auto-stopping solutions offer various customization options. Tags, scheduled triggers, and container-specific tools like KEDA enable tailored automation based on specific needs and deployment types. This flexibility ensures optimal resource utilization without compromising availability or workflow continuity.

A. Illustrative Examples:

[1]. Dev/Test Environments: Schedule auto-stopping for non-business hours using tagging and Lambda functions for EC2 instances or leverage KEDA for containerized deployments.

[2]. Production Clusters: Implement auto-stopping alongside autoscaling for specific nodes or deployments during predictable low-traffic periods.

Auto-stopping is critical for resource optimization and cost efficiency in cloud-native environments. Organizations can achieve substantial cost savings, streamline management processes, and minimize environmental impact by intelligently deactivating idle resources. By embracing auto-stopping alongside autoscaling, organizations can unlock a holistic approach to resource management, ensuring both performance and financial sustainability in the dynamic cloud landscape.

Benefits

[1]. Cost Savings: Auto-stopping offers substantial cost reductions in non-production and sporadically utilized environments by eliminating idle resource charges. Studies have shown potential cost savings ranging from 30% to 75%, depending on workload patterns.

[2]. Resource Efficiency: auto-stopping improves overall cluster resource utilization, enabling efficient allocation and freeing up resources for active workloads. This translates to a smaller infrastructure footprint and reduced environmental impact.

[3]. Simplified Management: Eliminating manual shutdowns reduces operational overhead and complexity, especially for large-scale deployments with fluctuating workloads.

Approaches

[1]. Lifecycle Hooks: Kubernetes deployments and jobs offer lifecycle hooks, allowing custom scripts to run before and after pod termination. These scripts can gracefully shut down applications, persist data, and perform cleanup tasks before complete deactivation.

[2]. Third-Party Tools: Specialized tools like cluster-api-provider-aws-controller leverage cloud provider APIs to automate node termination based on configurable criteria. These tools often offer advanced features like cost-optimized instance selection and integration with autoscaling mechanisms.

[3]. Kubernetes Custom Controllers: Custom controllers can be developed to monitor workloads and trigger resource deactivation based on specific policies. This approach provides maximum flexibility but requires deeper Kubernetes expertise and development effort.

Considerations

[1]. Application Health: Ensure auto-stopping gracefully shuts down applications and persists data to prevent unintended data loss or corruption. Careful evaluation of application behavior and recovery mechanisms is crucial.

[2]. Downtime Prevention: Configure auto-stopping thresholds and timeouts to avoid premature termination of critical workloads. Implement pre-shutdown checks to validate resource availability before deactivation.

[3]. Monitoring and Alerting: Monitor resource utilization and auto-stopping events to identify potential issues and ensure desired behavior. Consider setting up alerts for unexpected shutdowns or resource spikes.

Auto-stopping constitutes a powerful tool for optimizing resource utilization and reducing costs in Kubernetes. Organizations can leverage auto-stopping to achieve greater resource efficiency and cost consciousness within their cloud deployments by carefully evaluating potential approaches and addressing associated considerations.

Use Cases for Auto-stopping

Auto-stopping for Container Workloads

A. Auto-stopping Non-Production Container Workloads During Non-Business Hours

While Horizontal Pod Autoscaler (HPA) effectively adjusts resource allocation based on demand, it often maintains a minimum number of replicas even during periods of low activity. Auto-stopping can address this limitation, enabling complete resource shutdown when workloads are minimal.

Kubernetes Event-driven Autoscaling (KEDA) is a powerful tool for achieving this. KEDA introduces the concept of ScaledObjects, which define deployments and triggers for scaling based on various signals. Triggers can be configured using cron expressions, allowing for scheduled start and stop of deployments during specific non-business hours. For instance, a cron-based ScaledObject can be defined to set the minimum and maximum replica count to zero during off-peak hours, effectively suspending the deployment. You must set the max and min replicaset to zero to shut down your pods. You could write something similar to enable start at 8 a.m. PST to add pods.

```
apiVersion: KEDA.k8s.io/v1alpha1
kind: ScaledObject
metadata:
  name: {scaled-object-name}
spec:
scaleTargetRef:
    Name: {deployment-name}# must be in the same namespace as the ScaledObject
envSourceContainerName: {container-name}#Optional. Default: deployment.spec.template.spec.containers[0]
pollingInterval: 30# Optional. Default: 30 seconds
cooldownPeriod:  300# Optional. Default: 300 seconds
minReplicaCount: 0# Optional. Default: 0
maxReplicaCount: 0# Optional. Default: 0
triggers:
- type: cron
  metadata:
```

```
# Required
timezone: Americas/Chicago# The acceptable values would be a value from the IANA Time Zone Database.
    start: 018**1-5# at 6 pm every week day
    end: 08**1-5# at 8 am in the morning.
desiredReplicas: "10"
```

B. Procedure for auto-stopping Container Workloads (using KEDA)

KEDA offers a more streamlined approach to auto-stopping for containerized workloads managed by Kubernetes. As mentioned earlier, KEDA leverages ScaledObjects and triggers for workload scaling. By defining a ScaledObject for deployment and configuring a cron trigger with specific start and stop times, containerized workloads can be automatically suspended during designated off-peak hours. The minimum and maximum replica count within the ScaledObject definition should be set to zero to achieve complete resource shutdown.

```
apiVersion: keda.k8s.io/v1alpha1
kind: ScaledObject
metadata:
  name: {scaled-object-name}
spec:
scaleTargetRef:
deploymentName: {deployment-name}# must be in the same namespace as the ScaledObject
containerName: {container-name}#Optional. Default: deployment.spec.template.spec.containers[0]
pollingInterval: 30# Optional. Default: 30 seconds
cooldownPeriod:  300# Optional. Default: 300 seconds
minReplicaCount: 0# Optional. Default: 0
maxReplicaCount: 0# Optional. Default: 0
triggers:
- type: cron
  metadata:
# Required
timezone: Asia/Kolkata# The acceptable values would be a value from the IANA Time Zone Database.
    start: 018**1-5# at 6 pm PST every week day
    end: 08**1-5# at 8 am in the morning.
desiredReplicas: "10"
```

Auto-stopping for Virtual Machines (Non-containers)

A. Procedure for auto-stopping Non-Container Workloads (Virtual Machines)

auto-stopping can be implemented for non-containerized workloads deployed on cloud platforms like AWS using tagging and serverless functions. Resources can be tagged with a specific identifier (e.g., "shut-down") to indicate eligibility for auto-stopping. An AWS Lambda function can be designed to periodically check for resources with this tag and their current state. Based on a predefined schedule (managed by AWS EventBridge), the Lambda function can initiate stop and start actions for tagged EC2 instances.

Steps: This section details the steps for auto-stopping non-container workloads deployed on AWS using Lambda functions and Event Bridge.

[1]. Tagging Resources for auto-stopping

The first step involves tagging resources you want to auto-stop with a specific identifier (e.g., "shut down"). This tag clearly and efficiently distinguishes auto-stoppable resources.

[2]. Configuring IAM Permissions for the Lambda Role

An IAM role associated with the Lambda function requires specific permissions to interact with EC2 instances. Here's a breakdown of the required permissions:

A. **ec2:StartInstances, ec2:StopInstances:** These permissions allow the Lambda function to stop and start EC2 instances as needed.

B. **ec2:DescribeTags, ec2:DescribeInstances, ec2:DescribeInstanceTypes, ec2:DescribeInstance Status:** These permissions enable the Lambda function to view tags associated with EC2 instances, retrieve instance details (running/stopped status, instance types), facilitating the auto-stopping logic.

[3]. Implementing the Lambda Function

The Lambda function is the core component responsible for auto-stopping tagged EC2 instances. It iterates through all EC2 instances within your account and checks for the presence of the "shut-down" tag.

A. **Conditional Stopping and Starting:**

If the "shut-down" tag is set to true, the Lambda function executes code to stop the EC2 instance. This code triggers based on the scheduled event from EventBridge.

Conversely, the code to start the instance can be implemented and executed based on a separate EventBridge schedule or potentially another trigger, depending on your requirements. (An example for starting instances is provided in Step 4a.)

[4]. Scheduling auto-stopping Events with EventBridge

AWS EventBridge allows you to define schedules for triggering the Lambda function. This enables automated control over stopping and starting EC2 instances.

A. Example: Cron Expression for Shutdown

The provided cron expression (0 6 ? * 2-6 *) instructs EventBridge to trigger the shutdown Lambda function daily at 6:00 PM PST (Pacific Standard Time).

B. (Optional) Cron Expression for Startup

You can define a separate cron expression for starting instances. Your example (0 8 ? * 2-6 *) would trigger the startup Lambda function at 8:00 AM PST every weekday (Monday to Friday).

Note: This example uses cron expressions for scheduling. Event Bridge supports other scheduling options as well.

```
deflambda_handler(event, context):
 instances = ec2.instances.filter(Filters=[{'Name': 'instance-state-name', 'Values': ['stopped']},{'Name':
'tag:shutdown,'Values':[true]}])
for instance in instances:
    id=instance.id
    ec2.instances.filter(InstanceIds=[id]).start()
    print("the following instance ID has been started :- "+instance.id)
return"success"
```

Auto-stopping dev/test Environments during Non-business hours

Development and testing environments (Dev/Test) typically exhibit distinct usage patterns. While development activities primarily occur during business hours, occasional off-hours access might be required for automated testing or deployments. Auto-stopping these environments during non-business hours presents a significant cost-saving opportunity. Studies suggest potential cost reductions of at least 33% during weekdays by leveraging auto-stopping for Dev/Test environments.

Conclusion

Auto-stopping has emerged as a powerful technique for optimizing resource utilization and minimizing costs in cloud-native environments powered by Kubernetes. By strategically suspending idle resources during periods of low demand, organizations can achieve significant financial benefits without compromising application availability. This approach complements scaling mechanisms like HPA, enabling a more holistic and cost-effective resource management strategy.

A. Key Takeaways:

[1]. Auto-stopping intelligently deactivates underutilized resources, leading to substantial cost savings, particularly for non-production and sporadically used environments. Studies indicate potential cost reductions ranging from 30% to 75%.

[2]. Improved Resource Efficiency: auto-stopping optimizes overall cluster resource allocation, freeing up resources for active workloads and minimizing environmental impact through reduced energy consumption.

[3]. Streamlined Management: auto-stopping automates resource shutdowns and startups, simplifying management tasks and reducing operational overhead, especially for large-scale deployments with fluctuating workloads.

B. Looking Forward:

Auto-stopping will become increasingly important as cloud deployments develop to maintain performance and long-term financial viability. By using this methodology in conjunction with additional optimization strategies, enterprises can fully leverage the capabilities of their cloud infrastructure.

Reference

[1]. AWS Documentation. (October, 2020). "Automate Starting and Stopping AWS Instances". [Online]. Available:https://docs.aws.amazon.com/solutions/latest/instance-scheduler-on-aws/solution-overview.html

[2]. AWS Documentation. "Stop and start Amazon EC2 instances". (n.d.) [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/Stop_Start.html

[3]. AWS. "AWS Lambda". [Online]. Available: https://aws.amazon.com/lambda/

[4]. Kubernetes Documentation. "Horizontal Pod Autoscaling", (n.d.) [Online]. Available: https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale/

[5]. KEDA Documentation. "Scaling Deployments with KEDA", (n.d.) [Online]. Available: https://keda.sh/docs/1.5/concepts/scaling-deployments/